AMATH 482 Homework 4

Gonzalo Ferrandez Quinto

March 3, 2020

1 Introduction and Overview

On this writing I will be using different classification algorithms to classify handwritten numbers from the MNIST dataset. These handwritten numbers range from 0 to 9. The methods that I will be using are Linear Discrimination Analysis (LDA), Support vector machines (SVM) and decision tree classifiers (DTC). This data set contains both training and test data. I will using the the training set to create the classification algorithms and then use the test data to discuss the performance of the classification. I will also give the error rate on the test data.

In addition, it is worth noting that the MNIST data set also provides labels for each one of the images. I will be using these to know which is the desired output for training the algorithms and as way of calculating the error rate.

On the result section, I will be answering the following questions:

1. How many principal modes do I need to keep in the LDA analysis?
2. How does the data distribute on a 3D plot of the (2,3 and 5) columns of V?
3. Which 2 numbers are the most difficult and easiest to classify?
4. Can you do a 3 class classifier on this data base using LDA?
5. How do these compare to SVM and DTC?
6. How well do SVM and DCT classify all the numbers?

2 Theoretical Background

2.1 Linear Discrimination Analysis (LDA)

The idea behind the LDA is that for some given training data that includes a 2 class classification, the algorithm is trying to find a projection vector that maximizes the mean distance of the two data sets on this projection line. At the same time, it is also trying to minimize the spread of each class in this projection vector. Then we need to find some threshold on this projection line that divides both classes and reduces the number of misclassifications in the training data set. Therefore data points that project above and below this threshold will be classified in different classes. Before performing the LDA algorithm, I need to perform the SVD on the training data. I will first explain why we need the SVD.

2.1.1 SVD

The SVD is a tool that decomposes any matrix in a series of matrices multiplication that just involve rotations and stretches. In addition, the SVD is valid also for non-square matrices. Therefore any matrix of the form $m \times n$ can be decomposed by:

$$A = U\Sigma V^* \tag{1}$$

Here U is a $m \times m$ matrix and V^* is a $n \times n$ matrix, where both represent rotations. As these are rotation matrices, it will mean that they are unitary and therefore for both of them $V^{-1} = V^*$ (inverse equals the transpose). Also Σ is a real postive diagonal matrix of the form $m \times n$ which represents the stretch. In addition, its diagonal values are ordered in descending order from the highest to lowest. In the SVD basis then, the columns of the matrix U represents the principal directions of stretch that are occurring when the matrix A is applied as a linear transformation. In addition, the diagonal values of the matrix Σ represent how much a vector is being stretched in this direction.

2.1.2 The SVD in the context of LDA

To train the LDA algorithm, we need some differential features of the data that can make this linear discrimination possible. Therefore if the input data of both number images are reshaped in a vector and then all this vectors are stored in a matrix, the SVD can be performed on this matrix.

The interpretation of the Σ, U and V matrices are the following:

1. **Matrix** U The columns of this matrix represent the principal characteristics features of the data set. These will range by importance being the first columns more important. Also there will be one for each pixel in the input data.
2. **Matrix** Σ This matrix will tell us how important are each one of these principal features in characterizing the data. As the Σ matrix is diagonal, the diagonal elements correspond to the scaling of each one these characteristic features. It can also tell us how many different principal features of U should we keep, in order of properly classifying the data.
3. **Matrix** V The columns of this matrix will tell us how each image in the data contributes to each mode. Therefore there will be one column for each principal mode in U and one row per image. We will be using the matrix V to find differences in the data, by looking at how each numbers contributes differently to each mode.

This is the data that I will be introducing into the LDA algorithm. It will be the matrix SV' which by definition: $U'X = SV'$, where X is the data matrix. This can be understood as the projection of the data on each one of the principal components. We do not need to keep all the projections as we could keep a reduced number of projections as the first ones are more important.

2.1.3 LDA algorithm

After projecting the data on a reduced number of principal components. I will be separating the resultant matrix in the columns that represent the first class number and the second one. In this case, the columns of SV' (features X features)(features X image) represent each one of the images after the projection.

To quantify the inter-class spread, I will be using the value:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{2}$$

Here μ_i are column vectors that represent the average on each row of each class of the projected data. Therefore by calculating S_B, I am quantifying the variance across the classes.

To quantify spread with in each class, I will be using the value:

$$S_w = \sum_{j}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{3}$$

Here each x corresponds to a vector image and μ_j corresponds to the average over the rows of that class. This summation will be done for each image with the average of that class. After the total spread among both classes will be added.

To find the projection line we need to solve the equation:

$$W = \mathbf{argmax}\frac{W^T S_B W}{W^T S_w W} \tag{4}$$

Therefore this equation is trying to find a w that both maximizes the projected spread between classes and minimizes the projected spread on each class. It can be shown that the vector w satisfies the following generalized eigenvalue problem:

$$S_B W = \lambda S_w W \tag{5}$$

Therefore after finding W using the eigenvalue problem, we need to find a threshold in the line that reduces the number of classifications. Then any image projection lying on top or below of the line will be classified on a different class.

Finally when using this threshold on test data, we first need to project the images onto the principal components $U'x$. With U only having as many columns as features we are using on the LDA. Then we need to project $U'x$ onto the vector W. Then I will classifying this element depending on whether it falls over or under the threshold value.

2.2 Support vector machines and Decision tree classifiers

The idea behind the SVM, is to create a hyper plane that separates data clusters as much as it can. If you look at figure 2) the data is arranged forming clusters in this case the columns of V. Therefore it is possible to draw some plane that divides both clusters and classfies them. Therefore the input data on the SVM will then be (SV') which represents the columns of V, scaled by the importance (singular values.)

For decision tree classification, we are actually working with a neural network. The training data will then set the different weights between the neural network nodes that match an input with an output. therefore creating a predictive model.

3 Algorithm Implementation and Development

I will be exposing how I coded my answers for each one of the questions:

1. **principal modes** After reshaping each image on the training data as a vector on the data matrix, first I needed to perform a small analysis on how many modes should I keep. To do so I first used **svd()**. Then plotted the diagonal of the matrix **diag(Σ)** using **plot3()**. I will discuss my findings in the result section, but for the consequent LDA analysis, I will always be using 15 principal modes for the LDA.

2. **3D plot of columns of V** To perform this plot, I used plot3(). For this plot I used the columns of matrix V that was an output of the SVD of the previous step. I plotted the 60000 images of the training data using the 2nd, 3rd and 5th columns of V as coordinates for the plot. In this case,I used the labels of the MNIST dataset to plot each one of the clusters separately in order to color code and label them accordingly.

3. **2 class LDA** Before doing the LDA, I pre-processed the training data using the labels. I then stored each number in a structure such as train.ti for easier access. All the preprocessing is available in the pre.m script. Then given 2 input numbers, I grouped their data matrices in one matrix and performed the SVD on the whole data. Then on the projection SV', I separated the column classes and calculated their row mean averages using mean(). Then calculated the S_w matrix using a for loop and the S_b matrix using the averages. Then solved for the projection vector W using the eig() command. For which W is the eigenvector with the maximum absolute eigenvalue. Then after projecting each class on w, I ordered these using sort().Then I used a while loop to find a threshold value for which the same number of elements in each class are in the wrong side of threshold. I then also stored the number of errors in the training set.

To test the error rate on the test data, I iterated over all possible number combinations and performed the previous steps to train the LDA. Then after calling the number data matrices that correspond to that pair, I merged them into a single matrix and projected it onto the corresponding U matrix. Then I projected the result on the corresponding W vector. Finally, I checked if the data laid above or below the threshold. To calculate the error rate I then found the number of misclassifications in the projection using the LDA output and the MNIST labels. Then calculated the error rate by dividing this number over the total number test samples. The code is available in the sep2.m script which calls the classifygen.m fuction.

4. **3 class LDA** When performing the LDA on a 3 class problem, I first grouped two number data classes into one and then used the 2 class LDA for separating between one against the other two. Then I used the LDA algorithm again to classify between the first class which grouped two different numbers. After this two steps, the data was finally divided within the three classes. This can be done by using the class3.m script with the classifygen3.m function.

5. **SVM and DCT** When performing the SVM on a 2 class scenario, I will be using the fitcsvm() command. First from the pre-processed training data from the script pre.m, I will be calling the training data matrices that correspond for the 2 numbers I want to classify. Then I perform **svd()** command on the data. I will use then as the training input data (SV')/max(SV'). In this case, the matrix SV', will be of the form (features X features)(features X images).Therefore selecting how many principal modes we need to keep. We must also divide by the maximum value of the matrix to normalize the data and speed up the algorithm. Given this data and the labels, the algorithm is able to create a predicting model. I will also be using the fuction **resubLoss()** on the model created to find the training error.

   When applying this model on test data, I will be using the function **predict()**. Here the input is $(SV')_{test}/max(SV')_{training}$ Therefore taking the svd of the test data and trimming by the number of features. Then we need divide by the normalization factor of the training data for better results.

   When doing the SVM for classifying the whole dataset, I will be using instead **fitcecoc()** which is applied in the same way. Then I will use **confusionchart()** to show my results as a confusion matrix. The approach can be seen in the SVM code.

   To perform the DCT, I will be using the same code as as above but changing the training fuction to **fitctree()**

.

4 Computational Results

I will now be exposing my results for each one of the questions:

1. **principal modes** From figure 1), it can be seen that the first singular values have a fast drop and around the 15th singular value the slope is much less steep. That is why I used the 15 first principal modes to train the LDA algorithm. We cannot just keep the first one or two as the subsequent ones still have a big weight. Although the eyeball test is not rigorous the posterior error rates which was found corroborates the idea that this threshold was more than enough. Therefore I will be using a rank 15 approximation of the data.

2. **3D plot of columns of V** From figure 2), we can see how the different number images locate themselves in clusters which can easily be put apart. This reinforces the idea that the MNIST data is linearly separable, meaning that the different contributions to each one of the principal modes can be used to linearly classify the data using LDA.

3. **2 class LDA** After performing rank 15 LDA on all possible number combinations, I found that the numbers 1 and 0 where the ones that could most easily be classified. The training error for this set was of 0.24% and the test error rate was of 0.014%. Being this almost a perfect classification. Which makes sense as this are numbers that people tend to draw really differently.

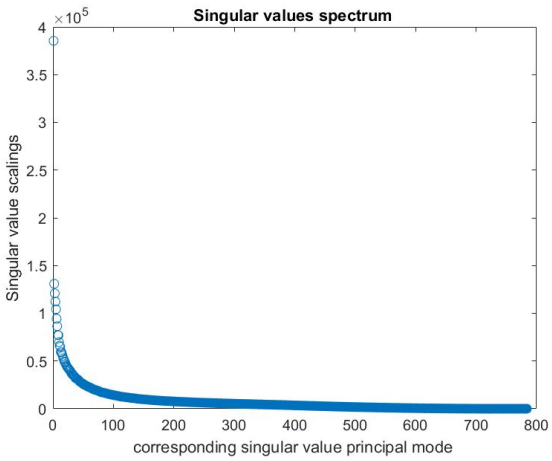

Figure 1: Singular value spectrum of the training data in the MNIST dataset

The number pair of 4 and 9 was the hardest to classify of all. This one had an error rate of 6.21% on the training data and an error of 6.78% on the test data. Although the error rate was much higher than in the previous case, it still has over 93% accuracy which is really impressive. This makes sense as many people write 9s and 4s in a similar manner. In addition, another pair of numbers that was difficult for classification where 3 and 5. It had an error rate of 7.5% on the training data and an error of 6.15% on the test data.

4. **3 class LDA** Using the methodology exposed in the algorithms section, I was able to classify data between 3 number classes. For example, classifying between 1,2 and 9 had an error rate on the test data of 21%. Another example is that classifying between 1,3 and 5 had an error rate of 27%. This could be explained by the higher similarities numbers have when merging their data to produce the first part of the LDA,

5. **SVM** When doing the SVM with 15 features on the 1/0 pair, I got 0.1% error on the training data. For the test data I got an error around 1%. For the 4/9 pair I got a much higher error with 6% trainning error and 44% test error. Although this error was highly reduced when using many more features.

   For the whole data, the confusion table can be seen in figure three. Number zeros had the highest accuracy and 1 the least. With ones confused with sevens. It also had 12% training error.

6. **DCT** This algorithm performed much better. Having in the 1/0 pair an error of 0.007% for training data and 0.4% for test data. In the 4/9 pair, this algorithm had a training error of 0.1% and a test error of 0.42% Also when classifying the whole data, it performed much better than the SVM. Having an overall better accuracy over all numbers. This can be seen in figure 4. It also has 4% trainning error.

5 Summary and Conclusions

During this whole project, the SVD was a very important tool in the classification of the data. It gave us the opportunity of determine the number of features we needed and also project into principal modes. The LDA is not a perfect methods and for low rank approximations, the DTC methods works much better.

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

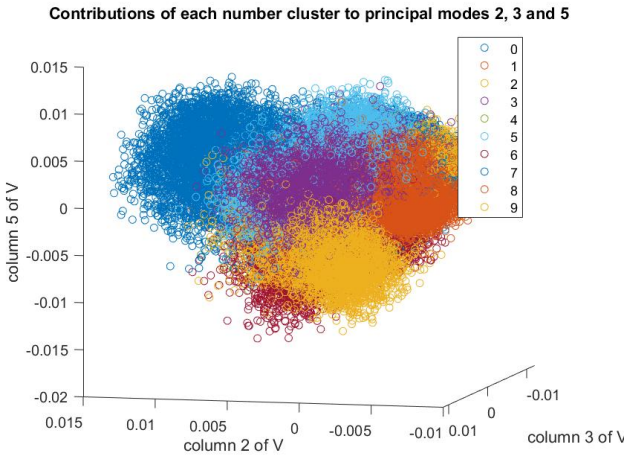

Figure 2: 3D data plot of the 2nd, 3rd and 5th columns of V. Which correspond to the contributions of each image to each one of the principal modes.

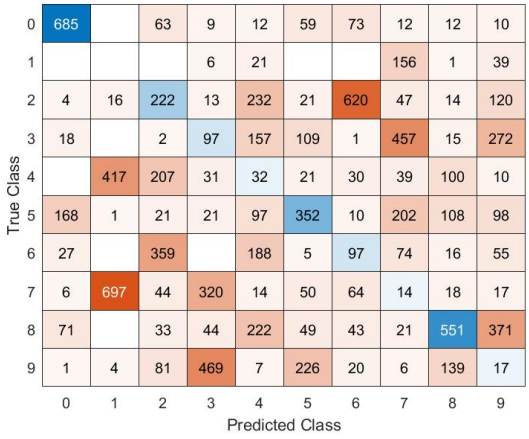

Figure 3: Confusion matrix of the MNIST test data using SVM classification

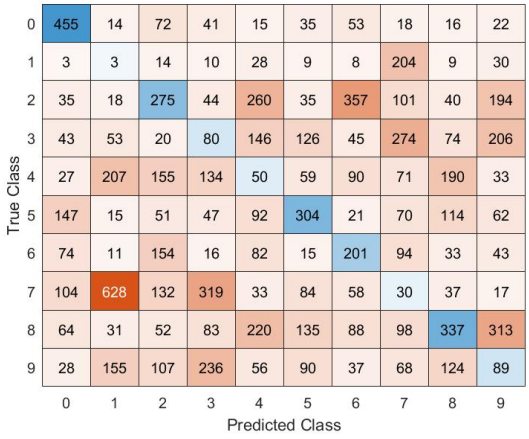

Figure 4: Confusion matrix of the MNIST test data using DTC classification

- `svd()` performs the SVD calculation on matrix.
- `fitctree()` creates neural tree model
- `fitcsvm()`creates hyperplane classification model
- `fitecoc()` returns the maximum value of an array
- `predict()` predicts out given input model and data
- `confusionchart()`outputs the confusion chart
- `plot3()` plots in 3d
- `eig()`solves eigenvalue equation

  `fin()`

Appendix B MATLAB Code

```
l = zeros(2,45)
l2 = zeros(2,45)
f = 15;
q = 0;
data1test = zeros(1,1);
data2test = zeros(1,1);

    for j=0:9
    for i=j:9
    if(j~=i)
    q = q+1;
    l2(:,q) = [i;j];

    result = classifygen(j,i,f,labels,data);
    U = result.u;
    w = result.w;
    s1 = size(test.("t"+string(j)),2);
    s2 = size(test.("t"+string(i)),2);
    testm = [test.("t"+string(j)),test.("t"+string(i))];
    av =  mean(testm,1);
 for x=1:size(testm,2)
     testm(:,x) = testm(:,x)-av(x);
 end
    testp = U(:,1:f)'*testm;
    testw = w'*testp;
    alt = result.alt;
    if(alt==false)
        res = testw > result.threshold;
    else
         res = testw < result.threshold;
    end
    correct = [logical(zeros(1,s1)), logical(ones(1,s2))];
    error = (res==correct);
    error = double(error);
    error = (size(error,2) - sum(error))/size(error,2);
    l(1,q) = error;
    l(2,q) = result.with;


    end
    end
end
```

Listing 1: separates 2 classes

```
number1 = 5
number2 = 1
number3 = 3
f =15;
result = classifygen3(number1,number2,number3,f,labels,data);
    U = result.u;
    w = result.w;
    de0 = size(test.("t"+string(number1)),2);
    de1 = size(test.("t"+string(number2)),2);
    d1 = [test.("t"+string(number1)),test.("t"+string(number2))];
    d2 = test.("t"+string(number3));
    s1 = size(d1,2);
    s2 = size(d2,2);
    testm = [d1,d2];
    av =  mean(testm,1);
 for x=1:size(testm,2)
     testm(:,x) = testm(:,x)-av(x);
 end
    testp = U(:,1:f)'*testm;
    testw = w'*testp;
    alt = result.alt;

    if(alt==false)
     res = testw > result.threshold;
    else
     res = testw < result.threshold;
    end


    correct = [logical(zeros(1,s1)), logical(ones(1,s2))];
    error = (res==correct);
    error = double(error);
    error = (size(error,2) - sum(error))/size(error,2);
    again = ones(1,s1+s2);
    index = zeros(1,1);
    for i=1:(s1+s2)
        if(res(i)==false)
            again(i)=0;
            index = [index,i];
        end
    end

    index = index(2:size(index,2))

    datag = [d1,d2];
    datag = datag(:,index);

     result = classifygen(number1,number2,f,labels,data);
    U = result.u;
    w = result.w;


    testm = datag;
    av =  mean(testm,1);
 for x=1:size(testm,2)
     testm(:,x) = testm(:,x)-av(x);
 end
    testp = U(:,1:f)'*testm;
    testw = w'*testp;
    alt = result.alt;
```

```
function result = classifygen(number1,number2,f,labels,data)
data1 = zeros(1,1);
data2 = zeros(1,1);


for i=1:60000
    if(labels(i)==number1)
        data1 = [data1,i];
    end
    if(labels(i)==number2)
        data2 = [data2,i];
    end
end
nb1 = size(data1,2);
nb2 = size(data2,2);
data1 = data1(2:nb1);
data2 = data2(2:nb2);

numb1data = data(:,data1);
numb2data = data(:,data2);

 tdata = [numb1data,numb2data];
 av =  mean(tdata,1);
 for i=1:size(tdata,2)
     tdata(:,i) = tdata(:,i)-av(i);
 end
 [U,S,V] = svd(tdata,'econ');

 numbers = S*V';
 n1 = numbers(1:f,1:nb1-1);
 n2 = numbers(1:f,nb1:nb2+nb1-2);

%% Calculate scatter matrices

m1 = mean(n1,2);
m2 = mean(n2,2);

Sw = 0; % within class variances
for k = 1:nb1-1
    Sw = Sw + (n1(:,k) - m1)*(n1(:,k) - m1)';
end
for k = 1:nb2-1
   Sw =  Sw + (n2(:,k) - m2)*(n2(:,k) - m2)';
end

Sb = (m1-m2)*(m1-m2)'; % between class

%% Find the best projection line

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

%% Project onto w

v1 = w'*n1;
v2 = w'*n2;

%% Make first number under threshold
```

```
function result = classifygen3(number1,number2,number3,f,labels,data)
data1 = zeros(1,1);
data2 = zeros(1,1);


for i=1:60000
    if(labels(i)==(number1||number2))
        data1 = [data1,i];
    end
    if(labels(i)==number3)
        data2 = [data2,i];
    end
end
nb1 = size(data1,2);
nb2 = size(data2,2);
data1 = data1(2:nb1);
data2 = data2(2:nb2);

numb1data = data(:,data1);
numb2data = data(:,data2);

 tdata = [numb1data,numb2data];
 av =  mean(tdata,1);
 for i=1:size(tdata,2)
     tdata(:,i) = tdata(:,i)-av(i);
 end
 [U,S,V] = svd(tdata,'econ');

 numbers = S*V';
 n1 = numbers(1:f,1:nb1-1);
 n2 = numbers(1:f,nb1:nb2+nb1-2);

%% Calculate scatter matrices

m1 = mean(n1,2);
m2 = mean(n2,2);

Sw = 0; % within class variances
for k = 1:nb1-1
    Sw = Sw + (n1(:,k) - m1)*(n1(:,k) - m1)';
end
for k = 1:nb2-1
   Sw =  Sw + (n2(:,k) - m2)*(n2(:,k) - m2)';
end

Sb = (m1-m2)*(m1-m2)'; % between class

%% Find the best projection line

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

%% Project onto w

v1 = w'*n1;
v2 = w'*n2;

%% Make first value under theshold
```

```
clear all
[images, labels] = mnist_parse('train-images.idx3-ubyte','train-labels.idx1-ubyte');

data = zeros(28*28,60000);
for i=1:60000
    im = images(:,:,i);
    im = im(:);
    data(:,i) = im;
end

[images2, labels2] = mnist_parse('t10k-images.idx3-ubyte','t10k-labels.idx1-ubyte');
datatest = zeros(28*28,10000);
for i=1:10000
    im = images2(:,:,i);
    im = im(:);
    datatest(:,i) = im;
end

 [U,S,V] = svd(data,'econ');
 d = diag(S);
 t = 1:784;
 dsum = zeros(1,784);
 summ = 0
 for i=1:784
 summ = summ + d(i);
 dsum(i) = summ;
 end
 plot(t,dsum/summ,'o')

 test = struct;
test.t0 = dividetest(0,datatest,labels2);
test.t1 = dividetest(1,datatest,labels2);
test.t2 = dividetest(2,datatest,labels2);
test.t3 = dividetest(3,datatest,labels2);
test.t4 = dividetest(4,datatest,labels2);
test.t5 = dividetest(5,datatest,labels2);
test.t6 = dividetest(6,datatest,labels2);
test.t7 = dividetest(7,datatest,labels2);
test.t8 = dividetest(8,datatest,labels2);
test.t9 = dividetest(9,datatest,labels2);

train = struct;
train.t0 = dividetest(0,data,labels);
train.t1 = dividetest(1,data,labels);
train.t2 = dividetest(2,data,labels);
train.t3 = dividetest(3,data,labels);
train.t4 = dividetest(4,data,labels);
train.t5 = dividetest(5,data,labels);
train.t6 = dividetest(6,data,labels);
train.t7 = dividetest(7,data,labels);
train.t8 = dividetest(8,data,labels);
train.t9 = dividetest(9,data,labels);



function result=dividetest(num,datatest,labels2)
    data1test = zeros(1,1);
    for m=1:size(datatest,2)
        if(labels2(m)==(num))
            data1test = [data1test,m];
        end
```

```
% % % SVM classifier with training data, labels and test set
% tott = 15
%
% s1 = size(train.t4,2);
% s2 = size(train.t9,2);
% data = [train.t4,train.t9];
% [U,S,V] = svd(data);
% v = S(1:tott,1:tott)*V(:,1:tott)';
% n = max(v(:));
% v = v/n;
%
% Mdl = fitcsvm(v',[zeros(1,s1),ones(1,s2)]');
%   testdata = [test.t4,test.t9];
%  [U,S,V] = svd(testdata,'econ');
% v = S(1:tott,1:tott)*V(:,1:tott)';
% v = v/n;
% errortrain = resubLoss(Mdl)
%   testlabels = predict(Mdl,v');
%  correct = [zeros(1,size(test.t4,2)),ones(1,size(test.t9,2))]
%  error = (logical(testlabels')==logical(correct));
%   error = double(error);
% errortest = (size(error,2) - sum(error))/size(error,2)*100;
%
% % %

% classify all numbers
tott = 15;
[U,S,V] = svd(data,'econ');
v = S(1:tott,1:tott)*V(:,1:tott)';
n = max(v(:));
v = v/n;

Mdl = fitcecoc(v',labels)
error = resubLoss(Mdl)

[U,S,V] = svd(datatest,'econ');
v = S(1:tott,1:tott)*V(:,1:tott)';
v = v/n;

error = resubLoss(Mdl)
[S,V,D] = svd(data,'econ')
testlabels = predict(Mdl,v')
C = confusionmat(testlabels,labels2);
confusionchart(C,{'0','1','2','3','4','5','6','7','8','9'})
cm.Title = 'Confusion matrix of SVM of MNIST dataset';
```

Listing 6: SVM and DTC code